Docker_Swarm_code

yeasy

Published
with GitBook



目錄

        xi. helpers.bash
        xii. port-filters.bats
        xiii. README.md
        xiv. resource_management.bats
        xv. run.sh
        xvi. swarm_id.bats
        xvii. test_runner.sh
    ii. regression
        i. run.sh
    iii. stress
        i. stress.bats

15. version
    i. version.go
16. CONTRIBUTING.md
17. Dockerfile
18. ISSUE-TRIAGE.md
19. LICENSE
20. logo.png
21. main.go
22. README.md
23. ROADMAP.md

Docker Swarm 源码分析

Swarm 是 Docker 官方推出的 Docker 容器集群管理项目。通过它，你可以在多个机器构建的集群上使用 Docker 容器，而无需关心具体被分配到哪台机器上。

本书将剖析 Swarm 组件的代码。

最新版本在线阅读：GitBook。

本书源码在 Github 上维护，欢迎参与：https://github.com/yeasy/Docker_Swarm_code。

感谢所有的 贡献者。

更新历史:

- V0.1: 2015-06-17
  - 完成基本结构。

参加步骤

- 在 GitHub 上 `fork` 到自己的仓库，如 `user/Docker_Swarm_code`，然后 `clone` 到本地，并设置用户信息。

```
$ git clone git@github.com:user/Docker_Swarm_code.git
$ cd Docker_Swarm_code
$ git config user.name "User"
$ git config user.email user@email.com
```

- 修改代码后提交，并推送到自己的仓库。

```
$ #do some change on the content
$ git commit -am "Fix issue #1: change helo to hello"
$ git push
```

- 在 GitHub 网站上提交 pull request。

- 定期使用项目仓库内容更新自己仓库内容。

```
$ git remote add upstream https://github.com/yeasy/Docker_Swarm_code
$ git fetch upstream
$ git checkout master
$ git rebase upstream/master
$ git push -f origin master
```

整体结构

源代码主要分为 7 个目录和若干文件： contrib，devstack，doc，etc，magnum，specs 和 tools。 除了这 7 个目录外，还包括一些说明文档、安装需求说明文件等。

contrib

主要包括一个 templates 目录，给出了实现一个 Bay Template 的例子。

devstack

包括将 Magnum 集成到 devstack 中的若干工具和文档。

doc

包括使用 Sphinx 生成文档的相关源码。

etc

跟服务和配置相关的文件，基本上该目录中内容在安装时会被复制到系统的/etc/ 目录下。

magnum

项目核心的代码实现都在这个目录下。 可以通过下面的命令来统计主要实现的核心代码量。

```
find magnum  -name "*.py" | xargs cat | wc -l
```

目前版本，约为 46k 行。

specs

项目提出时候的提案文档。

tools

一些相关的代码格式化检测、环境安装的脚本。

其它文档

- README.rst：介绍了项目的情况和连接。
- TESTING.rst：介绍如何进行开发后的测试。官方配置的 jenkins 当 gerrit 上有代码提交 review 的时候

会触发 tox 测试。实际上，OpenStack 中的项目使用 tox 来管理测试的虚拟环境，使用 testr 来管理运行测试案例的顺序。

- Dockerfile：生成一个 Docker 镜像，默认里面运行 Mangum-api 服务。

api

events.go

events_test.go

flusher.go

handlers.go

primary.go

README.md

replica.go

server.go

server_unix.go

server_windows.go

sorter.go

status.go

utils.go

utils_test.go

cli

命令行工具执行的入口。

cli.go

实现了一个 Run 方法。

生成一个新的应用 app，然后传递执行的命令和参数，调用应用中的 Run 方法。

```
if err := app.Run(os.Args); err != nil {
        log.Fatal(err)
    }
```

commands.go

定义支持的命令和参数格式。目前支持四个命令：

- create：创建一个集群；
- list：列出一个集群中的节点；
- manage：管理集群；
- join：加入一个集群。

```
var (
    commands = []cli.Command{
        {
            Name:      "create",
            ShortName: "c",
            Usage:     "Create a cluster",
            Action:    create,
        },
        {
            Name:      "list",
            ShortName: "l",
            Usage:     "List nodes in a cluster",
            Flags:     []cli.Flag{flTimeout},
            Action:    list,
        },
        {
            Name:      "manage",
            ShortName: "m",
            Usage:     "Manage a docker cluster",
            Flags: []cli.Flag{
                flStore,
                flStrategy, flFilter,
                flHosts,
                flLeaderElection, flManageAdvertise,
                flTLS, flTLSCaCert, flTLSCert, flTLSKey, flTLSVerify,
                flHeartBeat,
                flEnableCors,
                flCluster, flClusterOpt},
            Action: manage,
        },
        {
            Name:      "join",
            ShortName: "j",
            Usage:     "join a docker cluster",
            Flags:     []cli.Flag{flJoinAdvertise, flHeartBeat, flTTL},
            Action:    join,
        },
    }
)
```

create.go

create 方法，对应执行 create 命令。通过 token 来初始化一个 discovery，然后创建一个集群。

```
func create(c *cli.Context) {
    if len(c.Args()) != 0 {
        log.Fatalf("the `create` command takes no arguments. See '%s create --help'.", c.
    }
    discovery := &token.Discovery{}
    discovery.Initialize("", 0, 0)
    token, err := discovery.CreateCluster()
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(token)
}
```

flags.go

定义一些 flag 变量以及相关的处理方法。

```
var (
    flStore = cli.StringFlag{
        Name:  "rootdir",
        Value: homepath(".swarm"),
        Usage: "",
    }
    flJoinAdvertise = cli.StringFlag{
        Name:   "advertise, addr",
        Usage:  "Address of the Docker Engine joining the cluster. Swarm manager(s) MUST
        EnvVar: "SWARM_ADVERTISE",
    }
    flManageAdvertise = cli.StringFlag{
        Name:   "advertise, addr",
        Usage:  "Address of the swarm manager joining the cluster. Other swarm manager(s)
        EnvVar: "SWARM_ADVERTISE",
    }
    // hack for go vet
    flHostsValue = cli.StringSlice([]string{"tcp://127.0.0.1:2375"})

    flHosts = cli.StringSliceFlag{
        Name:   "host, H",
        Value:  &flHostsValue,
        Usage:  "ip/socket to listen on",
        EnvVar: "SWARM_HOST",
    }
    flHeartBeat = cli.StringFlag{
        Name:  "heartbeat",
        Value: "20s",
        Usage: "period between each heartbeat",
    }
    flTTL = cli.StringFlag{
        Name:  "ttl",
        Value: "60s",
        Usage: "sets the expiration of an ephemeral node",
    }
    flTimeout = cli.StringFlag{
        Name:  "timeout",
        Value: "10s",
        Usage: "timeout period",
    }
    flEnableCors = cli.BoolFlag{
        Name:  "api-enable-cors, cors",
        Usage: "enable CORS headers in the remote API",
    }
    flTLS = cli.BoolFlag{
        Name:  "tls",
        Usage: "use TLS; implied by --tlsverify=true",
    }
    flTLSCaCert = cli.StringFlag{
        Name:  "tlscacert",
        Usage: "trust only remotes providing a certificate signed by the CA given here",
    }
```

```
    flTLSCert = cli.StringFlag{
        Name:  "tlscert",
        Usage: "path to TLS certificate file",
    }
    flTLSKey = cli.StringFlag{
        Name:  "tlskey",
        Usage: "path to TLS key file",
    }
    flTLSVerify = cli.BoolFlag{
        Name:  "tlsverify",
        Usage: "use TLS and verify the remote",
    }
    flStrategy = cli.StringFlag{
        Name:  "strategy",
        Usage: "placement strategy to use [" + strings.Join(strategy.List(), ", ") + "]",
        Value: strategy.List()[0],
    }

    // hack for go vet
    flFilterValue = cli.StringSlice(filter.List())
    // DefaultFilterNumber is exported
    DefaultFilterNumber = len(flFilterValue)

    flFilter = cli.StringSliceFlag{
        Name:  "filter, f",
        Usage: "filter to use [" + strings.Join(filter.List(), ", ") + "]",
        Value: &flFilterValue,
    }

    flCluster = cli.StringFlag{
        Name:  "cluster-driver, c",
        Usage: "cluster driver to use [swarm, mesos-experimental]",
        Value: "swarm",
    }
    flClusterOpt = cli.StringSliceFlag{
        Name:  "cluster-opt",
        Usage: "cluster driver options",
        Value: &cli.StringSlice{},
    }

    flLeaderElection = cli.BoolFlag{
        Name:  "replication",
        Usage: "Enable Swarm manager replication",
    }
)
```

help.go

执行 help 命令的输出。

join.go

join 方法，对应执行 join 命令。

```
func join(c *cli.Context) {
    dflag := getDiscovery(c)
    if dflag == "" {
        log.Fatalf("discovery required to join a cluster. See '%s join --help'.", c.App.N
    }

    addr := c.String("advertise")
    if addr == "" {
        log.Fatal("missing mandatory --advertise flag")
    }
    if !checkAddrFormat(addr) {
        log.Fatal("--advertise should be of the form ip:port or hostname:port")
    }

    hb, err := time.ParseDuration(c.String("heartbeat"))
    if err != nil {
        log.Fatalf("invalid --heartbeat: %v", err)
    }
    if hb < 1*time.Second {
        log.Fatal("--heartbeat should be at least one second")
    }
    ttl, err := time.ParseDuration(c.String("ttl"))
    if err != nil {
        log.Fatalf("invalid --ttl: %v", err)
    }
    if ttl <= hb {
        log.Fatal("--ttl must be strictly superior to the heartbeat value")
    }
    d, err := discovery.New(dflag, hb, ttl)
    if err != nil {
        log.Fatal(err)
    }

    for {
        log.WithFields(log.Fields{"addr": addr, "discovery": dflag}).Infof("Registering o
        if err := d.Register(addr); err != nil {
            log.Error(err)
        }
        time.Sleep(hb)
    }
}
```

join_test.go

测试 join 命令，检查支持的地址的格式是否正确。

list.go

list 方法，响应 list 命令。

```
func list(c *cli.Context) {
    dflag := getDiscovery(c)
    if dflag == "" {
        log.Fatalf("discovery required to list a cluster. See '%s list --help'.", c.App.N
    }
    timeout, err := time.ParseDuration(c.String("timeout"))
    if err != nil {
        log.Fatalf("invalid --timeout: %v", err)
    }

    d, err := discovery.New(dflag, timeout, 0)
    if err != nil {
        log.Fatal(err)
    }

    ch, errCh := d.Watch(nil)
    select {
    case entries := <-ch:
        for _, entry := range entries {
            fmt.Println(entry)
        }
    case err := <-errCh:
        log.Fatal(err)
    case <-time.After(timeout):
        log.Fatal("Timed out")
    }
}
```

manage.go

manage 方法，响应 manage 命令。

```
sverify") && !c.IsSet("tlscacert") {
            log.Fatal("--tlscacert must be provided when using --tlsverify")
        }
        tlsConfig, err = loadTLSConfig(
            c.String("tlscacert"),
            c.String("tlscert"),
            c.String("tlskey"),
            c.Bool("tlsverify"))
        if err != nil {
            log.Fatal(err)
        }
    } else {
        // Otherwise, if neither --tls nor --tlsverify are specified, abort if
        // the other flags are passed as they will be ignored.
        if c.IsSet("tlscert") || c.IsSet("tlskey") || c.IsSet("tlscacert") {
            log.Fatal("--tlscert, --tlskey and --tlscacert require the use of either --tl
        }
    }

    store := state.NewStore(path.Join(c.String("rootdir"), "state"))
    if err := store.Initialize(); err != nil {
        log.Fatal(err)
    }

    uri := getDiscovery(c)
    if uri == "" {
        log.Fatalf("discovery required to manage a cluster. See '%s manage --help'.", c.A
    }
    discovery := createDiscovery(uri, c)
    s, err := strategy.New(c.String("strategy"))
    if err != nil {
        log.Fatal(err)
    }

    // see https://github.com/codegangsta/cli/issues/160
    names := c.StringSlice("filter")
    if c.IsSet("filter") || c.IsSet("f") {
        names = names[DefaultFilterNumber:]
    }
    fs, err := filter.New(names)
    if err != nil {
        log.Fatal(err)
    }

    sched := scheduler.New(s, fs)
    var cl cluster.Cluster
    switch c.String("cluster-driver") {
    case "mesos-experimental":
        log.Warn("WARNING: the mesos driver is currently experimental, use at your own ri
        cl, err = mesos.NewCluster(sched, store, tlsConfig, uri, c.StringSlice("cluster-o
    case "swarm":
        cl, err = swarm.NewCluster(sched, store, tlsConfig, discovery, c.StringSlice("clu
```

```
    default:
        log.Fatalf("unsupported cluster %q", c.String("cluster-driver"))
    }
    if err != nil {
        log.Fatal(err)
    }

    // see https://github.com/codegangsta/cli/issues/160
    hosts := c.StringSlice("host")
    if c.IsSet("host") || c.IsSet("H") {
        hosts = hosts[1:]
    }

    server := api.NewServer(hosts, tlsConfig)
    if c.Bool("replication") {
        addr := c.String("advertise")
        if addr == "" {
            log.Fatal("--advertise address must be provided when using --leader-election"
        }
        if !checkAddrFormat(addr) {
            log.Fatal("--advertise should be of the form ip:port or hostname:port")
        }

        setupReplication(c, cl, server, discovery, addr, tlsConfig)
    } else {
        server.SetHandler(api.NewPrimary(cl, tlsConfig, &statusHandler{cl, nil, nil}, c.B
    }

    log.Fatal(server.ListenAndServe())
}
```

cluster

mesos

queue

queue.go

queue_test.go

cluster.go

cluster_test.go

driver.go

offer_sorter.go

offer_sorter_test.go

README.md

slave.go

slave_test.go

task.go

task_test.go

utils.go

swarm

cluster.go

cluster_test.go

cluster.go

config.go

config_test.go

container.go

container_test.go

engine.go

engine_sorter.go

engine_sorter_test.go

engine_test.go

event.go

image.go

image_test.go

options.go

options_test.go

discovery

file

file.go

file_test.go

kv

kv

kv.go

kv_test.go

nodes

nodes.go

nodes_test.go

token

README.md

token.go

token_test.go

discovery.go

discovery_test.go

generator.go

generator_test.go

README.md

docs

api

swarm-api.md

images

virtual-box.png

scheduler

filter.md

strategy.md

discovery.md

Dockerfile

install-manual.md

install-w-machine.md

Makefile

release-notes.md

swarm-overview.md

Godeps

godep 是一套管理 Go 项目中依赖的工具，类似 Python 中通过 setup.py 或者 requirements.txt 来记录依赖信息。

它的使用十分简单。

保存当前项目使用的依赖信息：

```
$ godep save
```

基于保存的依赖库来构建项目

```
$ godep go install
```

或者

```
$ GOPATH=`godep path`:$GOPATH
$ go install
```

_workspace

godep 自动生成的存放依赖库的位置。

src

code.google.com

依赖的第三方包信息。

p.go-uuid.uuid。

github.com

依赖的第三方包信息。

- codegangsta
- coreos
- docker
- gogo
- golang
- gorilla
- hashicorp
- mesos
- samalba
- samuel
- Sirupsen
- stretchr

golang.org

依赖的第三方包信息。

x.net.context

Godeps.json

leadership

candidate.go

candidate_test.go

follower.go

follower_test.go

README.md

pkg

store

consul.go

consul_test.go

etcd.go

etcd_test.go

helpers.go

mock.go

README.md

store.go

store_test.go

zookeeper.go

zookeeper_test.go

scheduler

filter

affinity.go

affinity_test.go

constraint.go

constraint_test.go

dependency.go

dependency_test.go

expr.go

expr_test.go

filter.go

health.go

health_test.go

port.go

port_test.go

README.md

node

node.go

strategy

binpack.go

binpack_test.go

random.go

README.md

spread.go

spread_test.go

strategy.go

weighted_node.go

scheduler.go

script

demo

misc

script.sh

nsq

docker-compose.yml

redmon

docker-compose.yml

script.sh

coverage

travis_consul.sh

travis_etcd.sh

travis_zk.sh

validate-gofmt

state

state.go

store.go

store_test.go

test

integration

api

attach.bats

build.bats

commit.bats

cp.bats

create.bats

diff.bats

events.bats

exec.bats

export.bats

history.bats

images.bats

import.bats

info.bats

inspect.bats

kill.bats

load.bats

login.bats

logout.bats

logs.bats

pause.bats

port.bats

ps.bats

pull.bats

push.bats

rename.bats

restart.bats

rm.bats

rmi.bats

run.bats

save.bats

search.bats

start.bats

stats.bats

stop.bats

tag.bats

top.bats

unpause.bats

version.bats

wait.bats

discovery

consul.bats

discovery_helpers.bash

etcd.bats

file.bats

token.bats

zk.bats

mesos

api.bats

mesos_helpers.bash

testdata

build

affinities.bats

api_version.bats

cli.bats

constraints.bats

dependency.bats

Dockerfile

helpers.bash

port-filters.bats

README.md

resource_management.bats

run.sh

swarm_id.bats

test_runner.sh

regression

run.sh

stress

stress.bats

version

version.go

CONTRIBUTING.md

Dockerfile

ISSUE-TRIAGE.md

LICENSE

logo.png

main.go

执行 swarm 命令时候的入口。

```
package main

import (
    _ "github.com/docker/swarm/discovery/file"
    _ "github.com/docker/swarm/discovery/kv"
    _ "github.com/docker/swarm/discovery/nodes"
    _ "github.com/docker/swarm/discovery/token"

    "github.com/docker/swarm/cli"
)

func main() {
    cli.Run()
}
```

README.md

说明文件，介绍了项目定位（抽象多主机的容器资源为统一管理的接口），以及如何安装等信息。

ROADMAP.md

路线图，除了自带的资源管理后端，先支持 Mesos，将来支持 Kubernetes。